

# USB 4-Channel Video Capture Device

Model 2255

## Hardware and Software Manual

Ver.1.0.4 | June 2011

SENSORAY | embedded electronics



Designed and manufactured in the U.S.A.

SENSORAY | p. 503.684.8005 | email: [info@SENSORAY.com](mailto:info@SENSORAY.com) | [www.SENSORAY.com](http://www.SENSORAY.com)

7313 SW Tech Center Drive | Portland, OR 97203

# TABLE OF CONTENTS

LIMITED WARRANTY.....	3
SPECIAL HANDLING INSTRUCTIONS.....	4
INTRODUCTION.....	5
Feature Summary .....	5
Specifications.....	5
Operation basics.....	6
System Requirements.....	6
WINDOWS SOFTWARE.....	8
Installation.....	8
WINDOWS SDK REFERENCE.....	9
General Notes.....	9
Capture mode.....	9
Capture buffers.....	10
Functions Reference.....	11
LINUX SOFTWARE.....	18
Preamble(new V4L directory in SDK).....	18
Installation.....	18
LINUX SDK REFERENCE.....	20
General Notes.....	20
Capture mode.....	20
Capture buffers.....	21
Demo Application.....	21
Demo Application explanation.....	23
Driver Details (advanced).....	24
Functions Reference.....	25
REVISION HISTORY.....	32

## ***Limited warranty***

Sensoray Company, Incorporated (Sensoray) warrants the hardware to be free from defects in material and workmanship and perform to applicable published Sensoray specifications for two years from the date of shipment to purchaser. Sensoray will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

The warranty provided herein does not cover equipment subjected to abuse, misuse, accident, alteration, neglect, or unauthorized repair or installation. Sensoray shall have the right of final determination as to the existence and cause of defect.

As for items repaired or replaced under warranty, the warranty shall continue in effect for the remainder of the original warranty period, or for ninety days following date of shipment by Sensoray of the repaired or replaced part, whichever period is longer.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. Sensoray will pay the shipping costs of returning to the owner parts that are covered by warranty. A restocking charge of 25% of the product purchase price, or \$105, whichever is less, will be charged for returning a product to stock.

Sensoray believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, Sensoray reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult Sensoray if errors are suspected. In no event shall Sensoray be liable for any damages arising out of or related to this document or the information contained in it.

**EXCEPT AS SPECIFIED HEREIN, SENSORAY MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF SENSORAY SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. SENSORAY WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEROF.**

Third party brands, names and trademarks are the property of their respective owners.

## ***Special handling instructions***

The circuit board contains CMOS circuitry that is sensitive to Electrostatic Discharge (ESD).

Special care should be taken in handling, transporting, and installing circuit board to prevent ESD damage to the board. In particular:

- Do not remove the circuit board from its protective anti-static bag until you are ready to install the board into the enclosure.
- Handle the circuit board only at grounded, ESD protected stations.
- Remove power from the equipment before installing or removing the circuit board.

# Introduction

Model 2255 is a USB video capture device (frame grabber). It converts the signal from 4 analog video sources (NTSC or PAL) into a digital format (uncompressed or JPEG) and transfers the result to the host computer's RAM via a high-speed USB 2.0 interface. Model 2255 is powered through the USB port and does not require any external power supplies.

## Feature Summary

- Simultaneous capture from 4 composite video sources
- Total capture rate of 60 frames/sec from all channels for NTSC color video – 2 channels at 30 frames/sec each, or 4 channels at 15 frames/sec each
- Capture of JPEG compressed images
- Full frame rate capture on all channels in monochrome or scaled down modes
- Multiple output formats and resolutions
- Advanced deinterlacing of interlaced video eliminates motion artifacts
- Powered through USB
- Easy to use API; multiple units supported by the driver
- Windows and Linux support

## Specifications

Inputs	4 composite (BNC), 75 Ohm
Input video formats	NTSC (M), PAL (BDGHIMN)
Output formats	RGB packed (24 bits/pixel, bitmap compatible); YCrCb packed (16 bits/pixel, YUY2 compatible); YCrCb planar (16 bits/pixel, optimal for image processing); Y8 (8 bits/pixel, monochrome); JPEG <sup>1</sup>
Output resolutions	320x240, 640x480, 640x240 (NTSC) 352x288, 704x576, 704x288 (PAL) <sup>2</sup>
Capture rate	60 full size color frames per second (from all channels combined, NTSC) (50 for PAL). 120 full size monochrome or smaller size color frames per second (from all channels combined, NTSC) (100 for PAL). Same rates apply to JPEG capture.
Power	+5 V, 470 mA (through USB port)

---

<sup>1</sup> JPEG capture requires a special driver component which is sold separately. The regular software package includes an evaluation version of the driver which overlays a black bar across the captured image. Please contact Sensoray's sales for a full version of the driver.

<sup>2</sup> The NTSC capture formats are square pixel, PAL are not.

## Operation basics

Each channel of the 2255 behaves as a separate video capture device (frame grabber). It could be configured for a specific operating mode and capture rate independently of other channels.

The total capture rate from all 4 channels is limited to that of 2 full resolution color channels. It means that one could capture at full frame rate (30 frames/sec each channel for NTSC) from 2 channels, or at half frame rate (15 frames/sec each channel for NTSC) from 4 channels. Capture rate is limited by the amount of data transferred through the USB link, so reducing this amount increases the capture rate. For example, switching to monochrome mode allows capture at full frame rate from all 4 channels at full resolution.

In case of JPEG capture the frame rate is limited by the hardware capability to perform JPEG compression, but the resulting numbers turn out to be the same as with uncompressed video capture.

The 2255 could be configured for single frame or continuous capture. In single frame capture mode a command has to be sent to the board each time the frame is required. In continuous capture mode the frames are sent to the application at the selected capture rate. Choosing an appropriate mode allows balancing capture speed and CPU utilization.

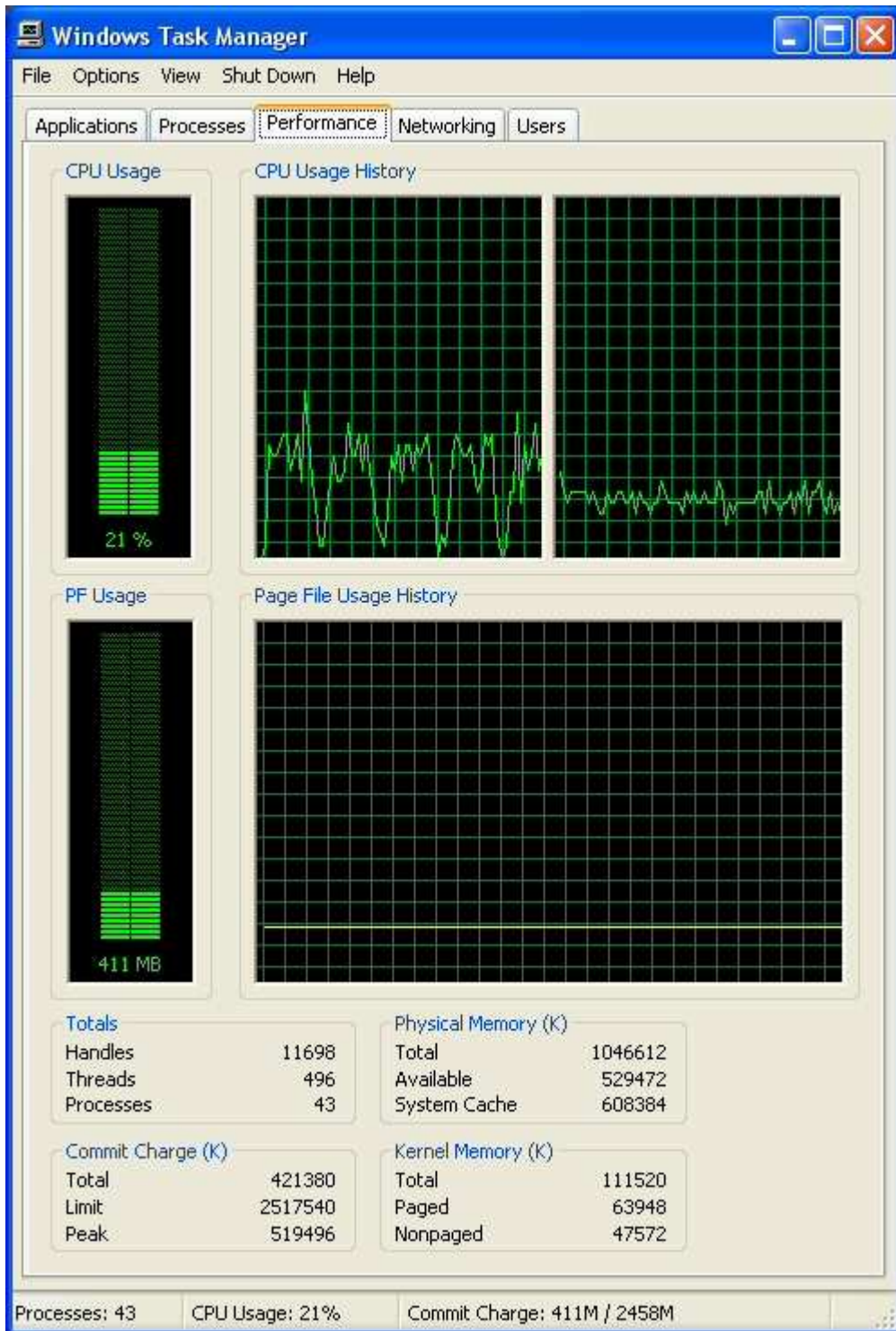
The 2255 allows capture of the interlaced frames (640x480 and 704x576 resolutions) or non-interlaced fields (all other resolutions). Interlaced images of fast moving objects, when viewed on the computer screen, display specific motion artifacts ("jaggies", "feathering") which result from the fact that two fields of one interlaced frame are captured by the camera sequentially in time. Alternatively, a capture mode is provided that recreates the second field using interpolation. That eliminates the motion artifacts at the expense of somewhat reduced vertical resolution.

## System Requirements

Transfers of large data amounts over the USB require more CPU attention than, for example, over the PCI bus using bus mastering devices. In addition to that in case of the 2255 image format conversions may be performed on the host side, depending on the output color format selected (only YCrCb planar and Y8 formats do not require any conversion).

Due to the factors mentioned above the host system has to satisfy the following minimum requirements: a Pentium III 700 MHz CPU with 512 MB of RAM. Sensoray recommends at least a Pentium IV-class 1 GHz CPU with 1 GB of RAM. A high-speed USB 2.0 port is required.

Below is a screen capture of the CPU usage graph from the system running a 2255 demo application capturing full size color NTSC video at 30 frames/sec from 2 channels in RGB format. The system is Dell Precision 380, Pentium D 2.8 GHz, 1GB of RAM. (Other applications were running concurrently, but were not using any significant CPU resources).



# **Windows Software**

## **Installation**

The following versions of Windows are supported: 2000, XP, Vista/7 (32 and 64 bits).

Two separate types of SDK are available: native and Direct Show. The drivers in those two types are not compatible, only one can be installed at a time. To switch to a different SDK type it is necessary to uninstall the old driver and install the new one.

Drivers and a demo application are available for download from the 2255 page at Sensoray's web site ([www.sensoray.com](http://www.sensoray.com)). A full-featured demo application allows capture and display of video on the host computer. Please note that no images are displayed by the demo application for the channel(s) configured for JPEG capture. JPEG images could be viewed by first saving those to the hard drive and using any JPEG viewer.

An included SDK can be used to integrate video capture into any application. The SDK allows maximum flexibility by providing an API for all the 2255's functions. The source code of the demo application is a suggested starting point for custom application development.

The downloaded file needs to be unzipped into a folder on the local drive prior to connecting the 2255 to the USB port. Run the setup program (setup.exe) from the distribution disk or folder. Software components, including a demo application with the source code, will be installed by default into the /Program Files/Sensoray/2255 folder. A shortcut to the demo application (app-2255-demo) will be installed in Start Menu under Programs/Sensoray/2255.

After the 2255 is connected to the USB port for the first time, a Windows "Found new hardware..." dialog appears. Select an option that allows specifying the driver location ("Install from a list or specific location"), click on "Browse" button and select the \Drivers subfolder of the Sensoray 2255 software disk or the folder where the downloaded file was unzipped to. If a "Windows Logo testing not passed" warning is displayed, click the "Continue anyway" button. The driver files will be copied and installed on your computer.

# Windows SDK Reference

## General Notes

### Capture mode

The operating mode of each of the 2255's capture channels is defined by the settings of the MODE2255 structure and is set by calling S2255\_SetMode function.

```
typedef struct {
    UINT32  format;    //input video format (NTSC, PAL)
    UINT32  scale;    //output video scale
    UINT32  color;    //output video color format
    UINT32  fdec;     //frame decimation
    UINT32  bright;   //brightness
    UINT32  contrast; //contrast
    UINT32  saturation; //saturation
    UINT32  hue;      //hue (NTSC only)
    UINT32  single;   //reserved (future use, leave as default)
} MODE2255;
```

#### format

Video format: FORMAT\_NTSC or FORMAT\_PAL.

#### scale

Image scale: SCALE\_4CIF5 - 640x480 (NTSC), 704x576 (PAL);  
SCALE\_2CIF5 - 640x240 (NTSC), 704x288 (PAL);  
SCALE\_1CIF5 - 320x240 (NTSC), 352x288 (PAL);  
SCALE\_4CIFSI - 640x480 (NTSC), 704x576 (PAL) with  
interpolation (deinterlacing);

#### color

Color format, one of the following:

COLOR\_YUVPL - YUV planar. The image data starts with an array of 1 byte Y component data for all the pixels, followed by the Cr data for all odd pixels, followed by the Cb data for all even pixels. Note that the amount of Y data is twice that of each color component.

COLOR\_YUVPK - YUV packed. The image data is presented in the following sequence: Y-Cr-Y-Cb-Y-Cr-Y-Cb, etc., where Y, Cb, and Cr are 1 byte each.

COLOR\_RGB - RGB, 3 bytes per pixel, Windows bitmap compatible (BGR sequence, image flipped vertically - bottom lines first).

COLOR\_Y8 - monochrome image, 1 byte per pixel.

COLOR\_JPG - color JPEG-compressed image. Compression quality (1..100, 100 being highest quality, largest file size) is passed in bits 8..15 of the color member. For all other color format settings contents of bits 8 through 15 is irrelevant.

fdec

Frame decimation setting. Must be one of the following: FDEC\_1 – capture every frame, FDEC\_2 – capture every second frame, FDEC\_3 – capture every 3<sup>rd</sup> frame, FDEC\_5 – capture every 5<sup>th</sup> frame. Other decimation values could be achieved by dropping frames in the application.

bright

Brightness level. Default is 0. Positive (in 2's complement form) values increase, and negative decrease brightness.

contrast

Contrast level. Default is 0x5c (unsigned). Minimum is 0x00, maximum is 0xff, which corresponds to a gain of approximately 4.

saturation

Saturation. Default is 0x80 (unsigned). Smaller values decrease, and larger values increase saturation.

hue

Hue (tint). Only works for NTSC. Default is 0, positive values give greenish tone, and negative values give purplish tone.

single

Unused.

### **Capture buffers**

The 2255 capture buffers are allocated by the application. This approach simplifies interfacing the 2255 to image processing software that uses its own buffers. The buffer's size (defined by `BUFFER.lpbmi[].biSizeImage`) has to be larger or equal to the size of the captured image. The SDK function `S2255_get_image_size` simplifies calculations of the necessary buffer size based on `S2255MODE` structure settings. For JPEG capture the buffer size has to be greater or equal than 81920 bytes (80 KB). Once allocated, the buffers must be registered with the driver with `S2255_RegBuffer` function before capture is started.

The buffers are allocated using a `BUFFER` type:

```
typedef struct {                //image buffer structure;
    ULONG dwFrames;             //number of frames in buffer;
    FRAME frame[SYS_FRAMES];    //array of FRAME structures;
    LPBITMAPINFO lpbmi[SYS_FRAMES]; //pointers to BITMAPINFO structure;
} BUFFER;
```

where `FRAME` type is defined as

```
typedef struct {                //frame structure;
    void *pdata;                //pointer to image data;
} FRAME;
```

Each BUFFER can contain up to SYS\_FRAMES (64) FRAME's. Using multiple FRAME's may be beneficial for preserving captured data in case of significant application latencies. On the other hand, when the primary function of the application is image display, having too many buffers will increase maximum display latency. For minimum display latency it is recommended to allocate 2 frames per buffer.

LPBITMAPINFO member is provided for convenience in those cases when images are captured in bitmap format. It points to the structure automatically created with a call to S2255\_RegBuffer function. Also, buffer size is returned in BUFFER.lpbmi[].biSizeImage member (including data size for JPEG capture mode).

## Functions Reference

```
S2255_DeviceOpen(int board, HANDLE *hdev);
```

Opens the 2255 device and acquires a handle.

Parameters

*board*

A 2255 index (0 for the first unit, 1 for the second unit, etc.).

*hdev*

a pointer to the variable accepting the handle value. All subsequent function calls use the handle to address the 2255.

Returns

0 on success.

```
S2255_DeviceClose(HANDLE hdev);
```

Must be called before application terminates for proper clean-up of the SDK and SDK objects.

Parameters

*hdev*

the handle value of the unit to be closed.

Returns

0 on success.

```
S2255_SetMode(HANDLE hdev, int channel, MODE2255 *mode);
```

Sets the operating mode for the selected channel (1-4) of the 2255.

Parameters

*hdev*

A handle to the 2255 device.

*channel*

channel number (1-4).

*mode*

a pointer to MODE2255 structure containing the desired operation mode (see s2255.h).

Returns

0 on success.

```
S2255_RegBuffer(HANDLE hdev, int channel, BUFFER *pBuf, UINT32 size);
```

Registers a buffer for a specific device/channel combination. The image buffers have to be allocated by the application software. The 2255 driver fills in the buffers with image data.

Parameters

*hdev*

A handle to the 2255 device.

*channel*

channel number (1-4).

*pBuf*

a pointer to BUFFER structure (see s2255.h).

*size*

requested buffer size.

Returns

0 on success.

```
S2255_DqBuf(HANDLE hdev, int channel, int frmnum);
```

Informs the driver that the application is releasing a frame back to the driver. Failure to do that may cause the driver to skip frames because of "buffer starvation".

Parameters

*hdev*

A handle to the 2255 device.

*channel*

channel number (1-4).

*frmnum*

frame number to dequeue (0 to SYS\_FRAMES - 1).

#### Returns

0 on success.

```
S2255_StartAcquire (HANDLE hdev, int channel, HANDLE *phevent);
```

Starts capture on the specified channel of the 2255.

#### Parameters

*hdev*

A handle to the 2255 device.

*channel*

channel number (1-4).

*phevent*

address of a variable accepting a handle to the event set by the driver each time a frame is ready.

#### Returns

0 on success.

```
S2255_StopAcquire(HANDLE hdev, int channel);
```

Stops capture on the specified channel of the 2255.

#### Parameters

*hdev*

A handle to the 2255 device.

*channel*

channel number (1-4).

#### Returns

0 on success.

```
S2255_GrabFrame (HANDLE hdev, int channel, unsigned char *pFrame,  
unsigned long size, unsigned long timeout, BITMAPINFO *pBMI);
```

Grabs a single frame from the board without the need to start acquisition (S2255\_StartAcquire) or register buffers (S2255\_RegBuffer). This method of capture is not recommended for continuous capture. Only use when not using S2255\_StartAcquire and S2255\_StopAcquire. Do not call this function after acquiring with S2255\_StartAcquire. Please see demo code for usage. Obsoletes S225\_GetFrame.

#### Parameters

*hdev*

A handle to the 2255 device.

*channel*

channel number (1-4).

*pFrame*

pointer to pre-allocated memory where image is written to (see size below).

*size*

size of pFrame. If in doubt, use S2255\_get\_image\_size to get required size.

*timeout*

timeout to wait for frame in milliseconds.

*pBMI*

pointer to bitmap info, which will be filled out on success.

Returns

0 on success.

```
S2255_GetImageSize( MODE2255 *mode );
```

Provides an easy way of determining the required buffer size based on the MODE2255 settings.

Parameters

*mode*

a pointer to MODE2255 structure.

Returns

Image size (bytes).

```
S2255_GetVidStatus(HDEVICE hdev, int channel, UINT32 *pStatus);
```

Returns the video input status on selected video channel.

Parameters

*hdev*

a handle to the 2255 device.

*channel*

channel number (1-4).

*pStatus*

a pointer to the current video status value. Bit [0]: a value of 1 indicates video signal present. Bit [8]: 0 for 60Hz signal (usually NTSC,) 1 for 50Hz signal (usually PAL). If Bit [0] is 0, then the value of bit [8] should be ignored.

Examples:

```
*pStatus = 0x0100 - no video signal present;  
*pStatus = 0x0001 - 60Hz video signal present;  
*pStatus = 0x0101 - 50Hz video signal present.
```

Returns

0 on success.

```
S2255_QueryBuf(HDEVICE hdev, int channel, int frmnum);
```

Queries the status of a given buffer.

Parameters

*hdev*

a handle to the 2255 device.

*channel*

channel number (1-4).

*frmnum*

Index of frame in question (starting from 0). Must be smaller than number of registered buffers.

Returns

Returns 0 if a frame buffer is unused. Returns 1 if a buffer is currently being filled (do not use buffer). Returns 2 if a buffer is ready (dequeue buffer after use). Returns -1 or negative value if failure occurs.

```
S2255_QueryBufExt(HDEVICE hdev, int channel, int frmnum, unsigned int *ts);
```

Queries the status of a given buffer and returns timestamp (in milliseconds) of frame.

Parameters

*hdev*

a handle to the 2255 device.

*channel*

channel number (1-4).

*frmnum*

Index of frame in question (starting from 0). Must be smaller than number of registered buffers.

*ts*

Timestamp in milliseconds when frame was captured on board. The timestamp is a running counter that is reset only when when board is powered up. The timestamp is 32 bits long and wraps around after an overflow. Resolution of timestamp is 1ms.

#### Returns

Returns 0 if a frame buffer is unused. Returns 1 if a buffer is currently being filled (do not use buffer). Returns 2 if a buffer is ready (dequeue buffer after use). Returns -1 or negative value if failure occurs.

```
S2255_GetSN(HDEVICE hdev, SN2255 *sn);
```

Returns the serial number on the board. Please note: Only boards manufactured after August 2010 have a serial number.

#### Parameters

*hdev*

a handle to the 2255 device.

*sn*

serial number structure (see s2255.h). Includes programming date.

```
S2255_Enumerate(int *pCount, DEVINFO2255 *pDevices);
```

Enumerates all 2255 boards in the system. If called with \*pCount equal to 0, the number of attached devices is set in \*pCount. If \*pCount != 0 and pDevices != NULL, then pDevices points to a list of at least \*pCount DEVINFO2255 structures which get filled in with board number and serial number information. Please see demo application for an example.

#### Parameters

*\*pCount*

Returns devices found in system (if called with \*pCount=0) or size of pDevices to fill with board information.

*pDevice*

array of at least \*pCount devices. If querying the number of devices (\*pCount = 0), pDevices may be NULL.

#### Returns

Returns 0 on success.

```
S2255_GetFirmware(HDEVICE hdev, UINT32 *usb, UINT32 *dsp);
```

Returns the board's firmware version. This function is informational only.

#### Parameters

*hdev*

a handle to the 2255 device.

*usb*

USB firmware version.

*dsp*

Current loaded DSP firmware version. 0 if unknown. Unpacking for display is as follows:  $x = dsp / 10000$ ,  $y = (dsp \% 10000) / 100$ ,  $z = (dsp \% 100)$ , version = x.y.z

#### Returns

Returns 0 on success.

```
S2255_fill_bmi_vbhelper(HDEVICE hdev, BITMAPINFO *pbmi, int chn, int frm);
```

Helper function for VB or VB.NET demo. This function is only intended for VB and not required for C programs. Fills the bitmap info structure with the current bitmap info structure for a particular frame and associated channel.

#### Parameters

*hdev*

a handle to the 2255 device.

*pbmi*

pointer to bitmap info structure

*channel*

channel number (1-4).

*frmnum*

Index of frame in question (starting from 0). Must be smaller than number of registered buffers.

#### Returns

Returns 0 on success.

# Linux Software

## Preamble(new V4L directory in SDK)

There are now 2 drivers available: the original driver and the V4L2 driver. Sensoray has worked to get a 2255 driver into the Linux kernel. It should be present in Linux kernel version 2.6.27. The V4L driver distributed in V1.1.8 of the Linux SDK is supported on Linux kernel versions 2.6.24 to the current 2.6.27 kernel. If you require the V4L driver on 2.6.18-2.6.23, backports of all current V4L modules/libraries are available from the Video for Linux working group at <http://www.linuxtv.org>. Sensoray supports only V4L2 as V4L1(version 1) has been marked obsolete by Linux developers.

There are advantages and disadvantages for each driver. It is up to the customer to decide which driver to use based on their application. Sensoray will support both drivers equally. The V4L driver, however, will not be supported below Kernel version 2.6.18. Please see the READMEs in the distributed SDK, if you choose to use the V4L driver. The rest of this manual after this Preamble describes the Sensoray 2255 driver and SDK. The Video for Linux API is well documented elsewhere.

Advantages of non-V4L driver(original):

In some earlier Linux distributions, V4L2 was not compiled by default into the kernel or as a module. With the non-V4L driver, it was not necessary to recompile the kernel to get things working. Additionally, some customers wanted an API similar to other Sensoray boards. Additionally, the V4L driver may not have all the features of the non-V4L driver. For instance, the non-V4L driver has frame decimation. The V4L API, at the time the driver was written, had frame decimation marked as an experimental feature.

Advantages of V4L driver:

The main advantage of the V4L driver over the non-V4L driver is standardized ioctl calls. There are many V4L2 applications available in the Linux community. The best known V4L application for display is xawtv.

## Installation

Drivers and a demo application for Linux 2.6 are available for download from the 2255 page on Sensoray's web site ([www.sensoray.com](http://www.sensoray.com)). A full-featured demo application allows capture and display of video on the host computer using XWindows.

An included SDK can be used to integrate video capture into any application. The SDK allows maximum flexibility by providing an API for all the 2255's functions. The source code of the demo application is a suggested starting point for custom application development. The full source code for the driver and middleware is available.

The Linux SDK requires a Linux distribution with development tools installed. It is assumed the user is familiar with Linux compilation and C programming.

The steps for installing and running the software follow:

- 1) `"tar xvzf drv-2255-linux.tgz"`.
- 2) `"cd drv-2255-linux/code"`
- 3) `make`
- 4) connect the 2255(s) to the host computer via USB.
- 5) Insert the driver `"insmod s2255.ko"`
- 6) Make the demo application with `"make -f make.demo"`.
- 7) Connect composite video channel to Channel 1.
- 8) Run the demo application. `"/s2255demo"`
- 9) Start streaming to XWindows on channel 1. `"Start 1"`
- 10) When done, stop the stream `"Stop 1"`.

# Linux SDK Reference

## General Notes

### Capture mode

The operating mode of each of the 2255's capture channels is defined by the settings of the MODE2255 structure and is set by calling S2255\_SetMode function.

```
typedef struct {
    UINT32  format;    //input video format (NTSC, PAL)
    UINT32  scale;    //output video scale
    UINT32  color;    //output video color format
    UINT32  fdec;     //frame decimation
    UINT32  bright;   //brightness
    UINT32  contrast; //contrast
    UINT32  saturation; //saturation
    UINT32  hue;      //hue (NTSC only)
    UINT32  single;   //unused
} MODE2255;
```

#### format

Video format: FORMAT\_NTSC or FORMAT\_PAL.

#### scale

Image scale: SCALE\_4CIF5 - 640x480 (NTSC), 704x576 (PAL);  
SCALE\_2CIF5 - 640x240 (NTSC), 704x288 (PAL);  
SCALE\_1CIF5 - 320x240 (NTSC), 352x288 (PAL);  
SCALE\_4CIFSI - 640x480 (NTSC), 704x576 (PAL) with  
interpolation (deinterlacing);

#### color

Color format, one of the following:

COLOR\_YUVPL - YUV planar. The image data starts with an array of 1 byte Y component data for all the pixels, followed by the Cr data for all odd pixels, followed by the Cb data for all even pixels. Note that the amount of Y data is twice that of each color component.

COLOR\_YUVPK - YUV packed. The image data is presented in the following sequence: Y-Cr-Y-Cb-Y-Cr-Y-Cb, etc., where Y, Cb, and Cr are 1 byte each.

COLOR\_RGB - RGB, 3 bytes per pixel, Windows bitmap compatible (BGR sequence, image flipped vertically - bottom lines first).

COLOR\_Y8 - monochrome image, 1 byte per pixel.

#### fdec

Frame decimation setting. Must be one of the following: FDEC\_1 - capture every frame, FDEC\_2 - capture every second frame, FDEC\_3 - capture every 3<sup>rd</sup> frame, FDEC\_5 -

capture every 5<sup>th</sup> frame. Other decimation values could be achieved by dropping frames in the application.

`bright`

Brightness level. Default is 0. Positive (in 2's complement form) values increase, and negative decrease brightness.

`contrast`

Contrast level. Default is 0x5c. Minimum is 0x00, maximum is 0xff, which corresponds to a gain of approximately 4.

`saturation`

Saturation. Default is 0x80. Smaller values decrease, and larger values increase saturation.

`hue`

Hue (tint). Only works for NTSC. Default is 0, positive values give greenish tone, and negative values give purplish tone.

`single`

Unused. Leave as default

### ***Capture buffers***

The Linux driver allocates the 2255 capture buffers in kernel space using virtual memory. The user can access these buffers using MMAP or by copying from the buffers (inefficient). The driver currently has 16 `SYS_FRAMES(s2255.h)`. When a start acquire is done, it starts a USB read pipe to continuously read into the buffers starting at frame index 0. It operates as a ring buffer. After frame index `SYS_FRAMES-1`, the driver will read into frame index 0 again. If frame index 0 hasn't been dequeued by the time the driver gets there again, frame index 0 will be skipped or the frame is dropped. The driver continuously reads the USB pipe to prevent any hardware/bus failures. It is up to the user application to dequeue the frames in a timely manner. If not, then frames are simply discarded.

### ***Demo Application***

**Capture to 2 channels simultaneously:**

- 1) `start 1`
- 2) `start 2`

**Capture to 4 simultaneous channels with full color(1/2 frame rate):**

- 1) `fdec 1 2` (channel 1 frame rate ½)
- 2) `fdec 2 2` (channel 2 frame rate ½)
- 3) `fdec 3 2`

- 4) fdec 4 2
- 5) start 1
- 6) start 2
- 7) start 3
- 8) start 4

**Capture 4 simultaneous channels, 1CIFs, full frame rate, full color:**

- 1) scale 1 3
- 2) scale 2 3
- 3) scale 3 3
- 4) scale 4 3
- 5) start 1
- 6) start 2
- 7) start 3
- 8) start 4

**Take snapshot of stream:**

- 1) start 1
- 2) snap file.bmp (save snapshot to file file.bmp)
- 3) stop 1 (optionally stop the stream)

**Run 2 2255 boards at the same time(channel 1):**

- 1) board 0
- 2) start 1
- 3) board 1
- 4) start 1

### **Demo Application explanation**

When the demo application opens, it tries to open all board instances(up to 8) in the system. It looks for each board in order and opens them. The first board is /dev/s2255\_0. The second board is /dev/s2255\_1.

To open the board, the function S2255\_DeviceOpen is called with the board number and a pointer to the device handle. The source code for S2255\_DeviceOpen is in the file s2255mid.c. For the Linux SDK, it opens a handle to the device with the standard call "open". This function also creates working buffers for possible future conversion functions.

After the call to S2255\_DeviceOpen, the mode for the 2255 must be set before starting any acquisition. This is done by the S2255\_SetMode function for each channel of the board being opened. The current supported color modes for the Linux SDK are COLOR\_YUVPL and COLOR\_Y8. COLOR\_RGB and COLOR\_YUVPK in s2255.h are for the Windows SDK only. Conversion to RGB is shown in the demo application to display to XWindows. The above calls are done in "board\_open(s2255demo.c)," which is called from "app\_init(s2255demo.c)".

The demo application then uses a command processor to determine which command to execute. Basic custom applications would just call whatever acquisition routines are required.

To start capturing on channel 1, the start\_stream function is called. This starts the io\_thread function that performs XWindows capture.

### **Capture thread**

The first task in the capture thread is to get the current image size. This is done with the function S2255\_get\_image\_size(&mode) with an argument of a pointer to the mode for the current channel.

Two methods of capture are possible. The preferred is using MMAP. The thread makes an ioctl call to S2255\_IOC\_REQBUF (In the future, this call may be moved to the middleware s2255mid.c, but the code will be the same) with a getbuf\_param\_t (s2255ioctl.h). After this call, mmap is done on the resulting buffer pointer to transform the kernel space(driver) buffer to a usable user space buffer area. S2255\_IOC\_REQBUF is called for each system frame of number NUMFRAMES.

Once MMAP is complete, acquisition is started with the function S2255\_StartAcquire. It has parameters for the device handle and the current channel. The third argument is for Windows only and should be ignored.

The next block of code creates a bitmap header and allocates work frames for the image manipulation. The 2255 currently captures in hardware in YUVPL(planar YUV) or Y8(monochrome) modes. To get RGB or YUY2, a transformation is required (image\_linux.c).

After the work buffers and DIB header are created, the XWindows window and image memory is created. The important information is ximage->data.

Following the XWindows creation is the acquisition loop while( !b->stop\_rq[idx] ) {}. The procedure is to wait for a frame(S2255\_wait\_frame), transform(planar\_422p\_to\_rgb565) and display the frame(XshmPutImage, XCopyArea, XSync) , and then dequeue the frame(S2255\_DQBUF). Optionally, if a snapshot is triggered, the acquisition loop will save the image to file with the planar\_to\_rgb8 and

save\_image\_uncompressed files. In the demo application, a snapshot is triggered with the command "snap filename.bmp".

After the acquisition loop is exited(with a "stop channel" command), the function S2255\_StopAcquire should be called. Finally, the user memory should be unmapped with munmap and the working buffers freed.

#### **Other demo functions**

- video brightness: "bright n" where n is -127 to 128 (default 0)
- video contrast: "contrast n" where n is 0 to 255 ( default 0x5c from s2255.h)
- video saturation: "vsat n" where n is 0 to 255 (default 0x80)
- video hue: "vhue n" where n is 0 to 255 (default 0)
- video system "vsystem n" where n=1 NTSC, n=2 PAL
- frame decimation "fdec n v" where n is the channel, v value(1 all frames, 2 every 2<sup>nd</sup>, 3 every 3<sup>rd</sup>, 5 every 5<sup>th</sup>)
- scale "scale n v" where n is the channel, v scale(1-4CIFs, 2-2CIFs, 3-1CIFs)

#### **Unsupported function(s) in demo application**

- Monochrome capture: Demo application does not currently demonstrate monochrome operation.

#### **Driver Details (advanced)**

This section is not meant for the average or even expert user. Most users can ignore all of this section. This section is only for OEMs who may want to port the driver to another OS such as VXWorks or QNX. This section assumes a basic knowledge of device drivers. The full details are not given here, just the main points about the USB pipes. The full source code for the Linux driver is available (to see the StartAcquire commands, etc...). Sensoray reserves the right to change any operation of the hardware/software.

The driver is composed of the following files:

s2255core.c : Core hardware operations. Should be mostly OS independent.

s2255mod.c: Module specific functions. OS dependent driver functions and IOCTLs.

s2255usb.c: Also OS dependent functions but specific to USB

size.c: size functions for various 2255 modes.

f2255usb.h: DSP firmware.

## Driver startup (probe function in Linux)

On driver startup, the driver creates a board handle in the core. This is done with the `S2255CORE_board_init` function and in Linux is called from `s2255_probe`, the probe function. `S2255CORE_board_init` initializes all the core specific variables and allocates USB pipe memory.

The probe function also saves the in and out endpoints of the USB. The 2255 has 2 endpoints. The out endpoint is for configuring and controlling the device. The in endpoint is for reading the buffers.

The probe function should download the firmware to the 2255. In Linux, this is done in chunks because the USB write functions must use `kmalloc'd` (continuous kernel space) memory. `Kmalloc'd` memory is a sparse resource and 128 KB is the maximum size in Linux 2.6. In `s2255_probe` (`s2255mod.c`) the firmware is downloaded via `S2255MOD_USB_WriteConfiguration` just before the probe function exits. If firmware download fails, probe should fail also as the device will be unusable.

## Open

When a file is opened, the `s2255_open` callback is called in the Linux driver. This function then calls `S2255CORE_board_open` in the core, which in turn starts the USB read pipe via `S2255MOD_USB_StartPipe` (`s2255usb.c`). The USB read pipe continuously reads data from the device. If no data is present, the read pipe just waits. When the device is closed, `S2255CORE_board_close` calls `S2255MOD_USB_StopReadPipe` which aborts or cancels any URB which was started at driver open. In Linux, `S2255_board_open` also creates all the system buffers for storing the frames.

## Functions Reference

```
S2255_DeviceOpen(int board, HANDLE *hdev);
```

Opens the 2255 device and acquires a handle.

### Parameters

*board*

A 2255 index (0 for the first unit, 1 for the second unit, etc.).

*hdev*

a pointer to the variable accepting the handle value. All subsequent function calls use the handle to address the 2255.

### Returns

0 on success.

```
S2255_DeviceClose(HANDLE hdev);
```

Must be called before application terminates for proper clean-up of the SDK and SDK objects.

Parameters

*hdev*

the handle value of the unit to be closed.

Returns

0 on success.

```
S2255_SetMode(HANDLE hdev, int channel, MODE2255 *mode);
```

Sets the operating mode for the selected channel (1-4) of the 2255.

Parameters

*hdev*

A handle to the 2255 device.

*channel*

channel number (1-4).

*mode*

a pointer to MODE2255 structure containing the desired operation mode (see s2255.h).

Returns

0 on success.

```
S2255_DQBUF(HANDLE hdev, int channel, int frmnum);
```

Informs the driver that the application is releasing a frame back to the driver. Failure to do that may cause the driver to skip frames because of "buffer starvation".

Parameters

*hdev*

A handle to the 2255 device.

*channel*

channel number (1-4).

*frmnum*

frame number to dequeue (0 to `SYS_FRAMES - 1`).

Returns

0 on success.

```
S2255_StartAcquire (HANDLE hdev, int channel, HANDLE *phevent);
```

Starts capture on the specified channel of the 2255.

Parameters

*hdev*

A handle to the 2255 device.

*channel*

channel number (1-4).

*phevent*

Unused. Set to NULL.

Returns

0 on success.

```
S2255_StopAcquire(HANDLE hdev, int channel);
```

Stops capture on the specified channel of the 2255.

Parameters

*hdev*

A handle to the 2255 device.

*channel*

channel number (1-4).

Returns

0 on success.

```
S2255_check_frame_poll(HDEVICE hdev, int channel, int *last, int *psize);
```

Returns 0 if frame is ready on given channel. Stores index of last frame in last. Non-blocking.

Parameters

*hdev*

Handle to device

*channel*

Channel to check(1-4)

*last*

Pointer to last frame

*size*

size of last frame (if capturing JPEGs)

Returns

0 if frame ready. Non-zero otherwise.

```
S2255_check_frame_poll_ts(HDEVICE hdev, int channel, int *last, int *psize, unsigned int *ts);
```

Returns 0 if frame is ready on given channel. Stores index of last frame in last. Non-blocking.

Parameters

*hdev*

Handle to device

*channel*

Channel to check(1-4)

*last*

Pointer to last frame

*size*

size of last frame (if capturing JPEGs)

*ts*

timestamp in ms

Returns

0 if frame ready. Non-zero otherwise.

```
S2255_check_frame_block(HDEVICE hdev, int channel, int timeout, int *last, int *psize);
```

Returns 0 if frame is ready on given channel. Stores index of last frame in last.

Parameters

*hdev*

Handle to device

*channel*

Channel to check(1-4)

*timeout*

Timeout in ms

*last*

Pointer to last frame

*size*

size of last frame (if capturing JPEGs)

#### Returns

0 if frame ready. Non-zero otherwise.

```
S2255_check_frame_block_ts(HDEVICE hdev, int channel, int timeout, int *last, int *psize, unsigned int *ts);
```

Returns 0 if frame is ready on given channel. Stores index of last frame in last and returns hardware timestamp.

#### Parameters

*hdev*

Handle to device

*channel*

Channel to check(1-4)

*timeout*

Timeout in ms

*last*

Pointer to last frame

*size*

size of last frame (if capturing JPEGs)

*ts*

timestamp in ms

#### Returns

0 if frame ready. Non-zero otherwise.

```
S2255_get_buffer(HDEVICE hdev, int channel, int frame, char *pdata, int size);
```

Copies the buffer specified to pdata. Use if mmap not used (double buffering).

#### Parameters

*hdev*

Handle to device

*channel*

Channel(1-4)

*frame*

index of frame

*pdata*

pointer to data

*size*

size of pdata

Returns

0 on success. Non-zero otherwise.

```
S2255_get_image_size( MODE2255 *mode );
```

Provides an easy way of determining the required buffer size based on the MODE2255 settings.

Parameters

*mode*

a pointer to MODE2255 structure.

Returns

Image size (bytes).

```
S2255_get_vid_status(HDEVICE hdev, int channel, UINT32 *pStatus);
```

Returns the video input status on selected video channel.

Parameters

*hdev*

a handle to the 2255 device.

*channel*

channel number (1-4).

*pStatus*

a pointer to the current video status value. Bit [0]: a value of 1 indicates video signal present. Bit [8]: 0 for 60Hz signal (usually NTSC,) 1 for 50Hz signal (usually PAL). If Bit [0] is 0, then the value of bit [8] should be ignored.

Examples:

\*pStatus = 0x0100 - no video signal present;

\*pStatus = 0x0001 - 60Hz video signal present;

\*pStatus = 0x0101 - 50Hz video signal present.

Returns

0 on success.

```
S2255_GetSN(HDEVICE hdev, SN2255 *sn);
```

Returns the serial number on the board. Please note: Only boards manufactured after August 2010 have a serial number.

#### Parameters

*hdev*

a handle to the 2255 device.

*sn*

serial number structure (see s2255.h). Includes programming date.

## ***Revision history***

Version	Notes
1.0.0, October 2010	Initial release.
1.0.1, January 2011	General cleanup.
1.0.2, February 2011	References to SECAM removed.
1.0.3, May 2011	Supported versions of Windows clarified.
1.0.4, June 2011	Info on <code>S2255_fill_bmi_vbhelper</code> function added. Typo fixed.